**Academic Services Group**

Creating the functional and nonfunctional requirements during the requirements engineering helped before beginning the implementation because these requirements described all the functionality we needed to include in our application. This helped us create prototypes of our application before we began implementing it. The requirements and prototype also gave us material so we could discuss the application with the customer in a way that was easy for them to understand. Our team believes that going through the UML modeling before implementation was helpful. The modeling helped us figure out the classes we would need, how the classes would interact, and how we would interact with the various APIs we would use in the application. However, this extensive modeling did not prepare us for some of the roadblocks that occurred during implementation. For example, we planned on sending JSON requests to one of our APIs to get species information; however, we discovered it would be easier and we would get better information if we sent a XML request to the specific API. This caused our implementation to deviate from our UML models. Also, since we are creating an android application, a major component of this first sprint was creating the layouts for the application. Our UML models did not help us with this aspect of implementation, but our prototypes did.

Our team thinks going through the requirements engineering phase will create a more stable product. One reason is that the agreed upon functional and nonfunctional requirements specify all the functionality of our application. This does not allow any feature creep which would make us rush to add more functionality. Functionality hastily added to an application is usually full of defects so creating an agreed upon document prevents the client from requesting the addition of any major functionality. Letting the client review and agree to the requirements document will help us create an application that will meet their needs. Creating the sequence and class models helped us discover different helper classes that could be used by multiple components of our application. For example, in this sprint, we created a helper function that condenses some of the information from one of the APIs into an accessible format. Overall, UML modeling helped us reduce code duplication and helped us modularize our code. Unfortunately, the benefits of UML modeling did not justify the amount of time spent on them.

If our team were to start another large project, we would create prototypes and list all the requirements of the application. Both of these components give you something to discuss with the customer in a way that is easy for them to understand. It is important that the customer understands your application before they approve development; otherwise, they might end up with a product that would not meet their needs. However, in the future, our team would not go through the extensive UML modeling. We would probably create a rough context/architectural model to get an idea of the components in our application and how they would interact, but we probably would not create the sequence and class diagrams. These diagrams were time consuming to create, and they did not help us with some of the roadblocks we encountered during implementation.

We converted fourteen of our requirements into user stories. We believe that this was enough for our sprint because we completed several of these user stories and made good progress on the ones we did not complete. The search for species user stories ended up being more difficult than we expected, but we made progress on these difficult stories. To convert the requirements to user stories, we used the “As a \_\_\_, I want a \_\_\_ so that I can \_\_\_” format. We first converted some of our most important functional requirements. These were the “The application shall…” user stories. This would allow us to begin development on our most important features in the first sprint. All team members were present during the conversion of the requirements. We all met up one evening to carry out this task.

Playing “Planning Poker” was not very helpful with story point estimation because all team members agreed on the story points for each story with a deviation of roughly 2 points. There were no stories in which the team as a whole could not compromise on a story point number.

For our daily scrums, we either met in person or communicated through video chat using Google Hangouts. When we held our scrums through video chat, we usually agreed to have them at 5:30 P.M. The times we meet in person for our scrums fluctuated, but we usually met after class around 11:00 A.M. or during the evening. Our scrums usually lasted about 10 to 20 minutes; however, some lasted longer than this. For example, one of our scrums involved us debugging GitHub, and it lasted nearly an hour through video chat.

The estimated velocity for our sprint was 39.5. The number of story points actually accomplished was 35.5. We did not finish either of the stories to search by species (whether common or scientific name); each of the two stories were worth six story points. We grabbed one additional story from the product backlog during our sprint, which was the creation of the back button; this story was worth two story points. This story was key to allowing us to move between pages, and it should have initially been in our sprint backlog. The overall workload was sufficient for everyone. None of our team members finished all of their stories before the sprint was completed. At the end of the sprint, Nathan Cooley, Brad Ewing, Kaleigh Key, Christopher Hebert, and Michael Towns had completed all of their stories. Shelby Pace did not complete her two stories for searching by species; however, these were prominent stories which are a challenging aspect of the app’s functionality. Shelby spent a great deal of time researching the APIs and learning how to best retrieve and display the information for our application. We each spent a substantial amount of time learning how to use GitHub within Android Studio and fixing some of the issues discovered during this process. Additionally, five of the six members in our group had to spend a lot of time getting to know Android and learning to code layouts in XML. There were additional tasks completed during this sprint that could have been stories but were not thought about as necessary beforehand (such as the need for error handling and the creation of certain aspects of our layouts). Our product is potentially shippable. It functions by allowing the user to type in the name of a state and select “Go,” populating a list of the species in the state. It also allows the user to switch to a help and settings page, which are not incredibly useful as of yet, but they will eventually provide value to the user. What is important is that they are present and formatted to a degree.

**Story points:**

**Nathan Cooley** Set out to accomplish: 4 Actually accomplished: 6

**Brad Ewing** Set out to accomplish: 7 Actually accomplished: 7

**Chris Hebert** Set out to accomplish: 5 Actually accomplished: 5

**Kaleigh Key** Set out to accomplish: 4.5 Actually accomplished: 4.5

**Shelby Pace** Set out to accomplish: 6 Actually accomplished: 0

**Michael Towns** Set out to accomplish: 13 Actually accomplished: 13

**Commits:**

**Nathan Cooley** Total meaningful commits: 5

**Brad Ewing** Total meaningful commits: 10

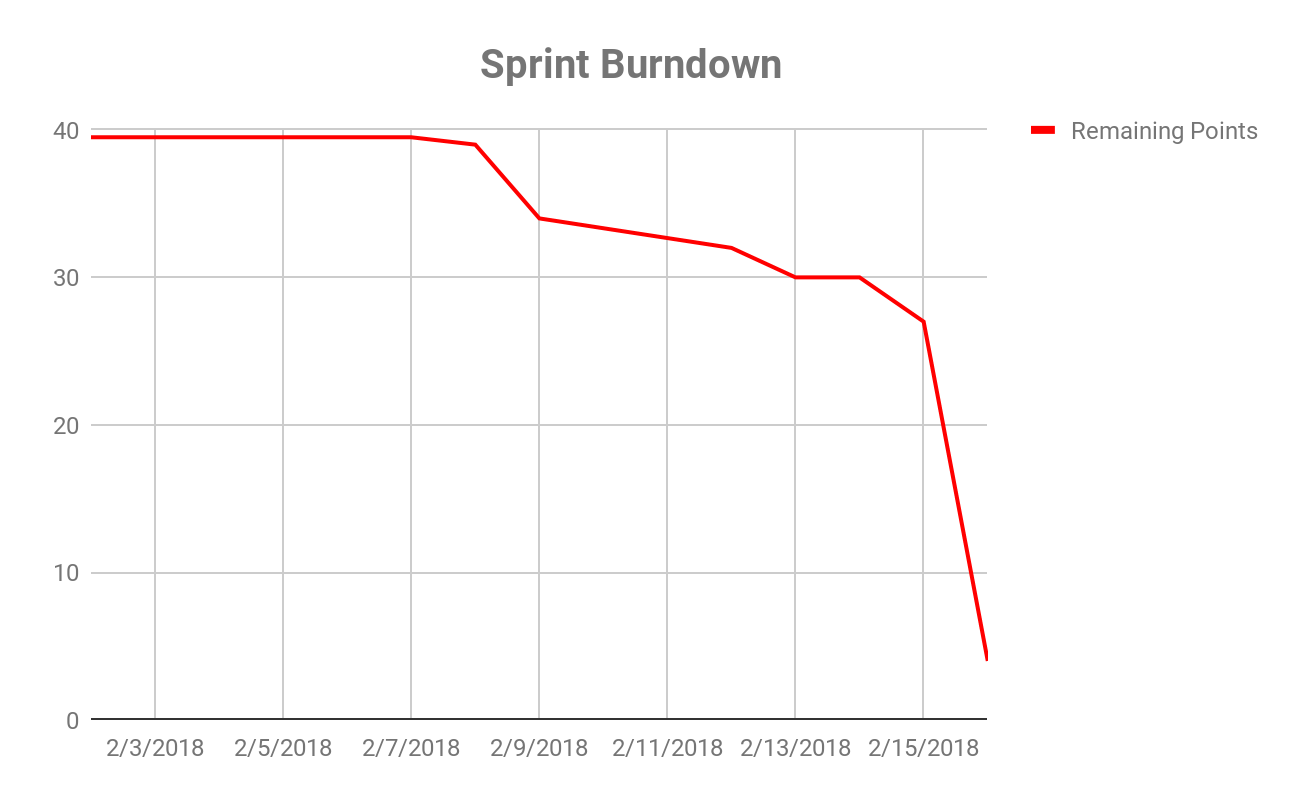
**Chris Hebert** Total meaningful commits: 12

**Kaleigh Key** Total meaningful commits: 11

**Shelby Pace** Total meaningful commits: 6

**Michael Towns** Total meaningful commits: 16

For the next sprint, we would keep a similar number of story points (possible increasing the number slightly). Even though we did not finish all user stories, most of our time was spent learning how Android Studio, Git, and Java work. We should be able to accomplish more story points in the future than in this initial sprint. For our next sprint, we are going to create more detailed stories, which are less vague. Also, we are going to do a better job communicating with one another about what we have accomplished or decided to additionally complete.

**Burndown Chart and Tables:**

|  |  |
| --- | --- |
| **Date** | **Remaining Points** |
| 2/2/18 | 39.5 |
| 2/5/18 | 39.5 |
| 2/6/18 | 39.5 |
| 2/7/18 | 39.5 |
| 2/8/18 | 39 |
| 2/9/18 | 34 |
| 2/12/18 | 32 |
| 2/13/18 | 30 |
| 2/14/18 | 30 |
| 2/15/18 | 27 |
| 2/16/18 | 4 |
| **Velocity** | 35.5 |

|  |  |
| --- | --- |
| **Date** | **Stories Moved to Done** |
| 2/8/18 | (0.5) As a developer, I want the application to open the search page (home screen) whenever it is first opened so that the user has a consistent home screen. [1] |
| 2/9/18 | (3) As a user, I want to view a help page so that I know how to use the application. [5] |
| (2) As a user, I want to open the settings page with a button so that I can view the settings page. [3] |
| 2/12/18 | (2) As a developer, I want a splash screen that is shown while the search page (home screen) loads so that the user knows the application is loading. [2] |
| 2/13/18 | (2) As a user, I want to open the help page with a button that is located on every page so that I can view the help page. [5] |
| 2/15/18 | (3) As a user, I want to view a settings page so that I can change the settings in the application. [5] |
| 2/16/18 | (2) As a user, I want to have a back button on each page so that I can return to the previous page. [2] |
| (3) As a developer, I want each page to have a basic design so that I can move from page to page. [2] |
| (1) As a user, I want to be able to click the banner from every page so that I can always return to the search page. [3] |
| (2) As a developer, I want to a search by state so information is returned about the species. [5] |
| (2) As a developer, I want to a search by county so information is returned about the species. [8] |
| (13) As a developer, I want to pull relevant info from the APIs, based on the scientific name, so that I have a way to display the data consistently. [21] |